NORTH DAKOTA STATE UNIVERSITY

REM Sleep Monitor II

ECE 405: Design III
SD1301

Ryan Klemisch
James Parrow
Alex Jenkins
Britta Olson

This document contains technical information pertaining to the REM Sleep Monitor Design Project from spring and fall 2013 semesters. Included in this report are diagrams, descriptions, schematics, software flow charts, problems encountered, lessons learned, and photos of the device.

Table of Contents

Background

The REM sleep monitor detects REM sleep using the electrooculogram (EOG). The EOG is a potential difference between the back and the front of the human eye. When the eye looks left and then right, there is a change in the polarity between the retina and the cornea. The EOG is very small, generally only a couple microvolts. The signal is obtained by wearing a headband with built in electrodes. The device reads the EOG signal and determines whether or not the user is in REM sleep. The device subtly alerts (low frequency/low volume) the user via a speaker once they've entered REM sleep and directly after exiting REM an annoying (high frequency/high volume) alert is heard. The intent of the device is to induce lucid dreaming allowing the user to be aware that they are currently in a dream. This type of device has the potential to help better understand the science of dreaming, specifically lucid dreaming. Currently, REM sleep devices are very expensive; the goal is to design a cost effective device that is able to be replicated by the general public. A "How to Guide" is now available for the public to make a proto-board based device at home. The proto-board is to be populated on a breadboard. An official PCB version of the device was fabricated as well.

Requirements

This device has to meet the following requirements in order to satisfy the needs of Professor Lima:

•Ability to be tested and be verified

•Accurately detect REM sleep

•Send cue to user during REM and after REM

•Transmit information via Bluetooth to computer to be stored and analyzed

•Battery operated

•Compact and non-invasive

Headband and Electrodes

In order to obtain a clean EOG signal, the headband has to be worn tightly and the electrodes must have minimal impedance. The headband and electrodes also have to be easily replicated. For this design, one will only need a hot glue gun, soldering iron and scissors. The headband is made of 1" x 25" polyester wedded strap, seen below in white. The black connector is a ladder lock side slider release buckle. This allows for easy adjustment and a tight fit. The electrodes are made of eight-micrometer thick copper, backed with a thicker polymer film. The pads are 1" x 1" squares with wires soldered to the center. The electrodes and the speaker are connected to the webbing via sliders made of ribbon. This allows any user to adjust the headband and electrodes to fit their head size.

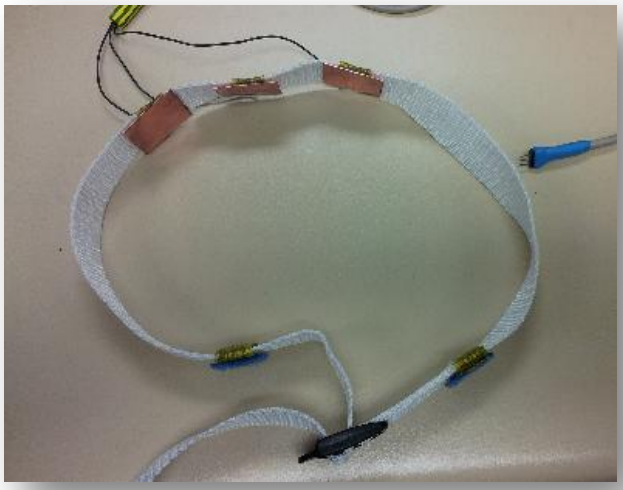


Figure 1: Headband with electrodes

Speaker:

A piezoelectric speaker was used to alert the user. Speakers were chosen as hearing is the only human sense that is not completely shut off while sleeping. Since humans are continuously, subconsciously, aware of sounds throughout the night, sending an audible cue would be less likely to actually wake the user while trying to induce lucid dreaming. The alert signal is first sent after the preset wait time has been surpassed. The initial signal is a very subtle 250Hz signal. With practice, the user can train to recognize the noise while dreaming and become aware that they are dreaming, but not wake up. The device will continue scanning to ensure the user is still in REM. Once the device no longer detects REM (i.e. the user has exited REM), the second alert signal will be activated. This is a very obnoxious 2,500Hz sound, which will quickly wake the user. This allows the user to better remember their dream and make more sense of the dreaming experience.

Figure 2: Piezoelectric Speaker

Hardware

Block Diagram:

This device acquires the user's EOG signal via **electrodes** placed on the forehead. The **high-pass filter** following the electrodes block DC offsets. The EOG signal is then fed into an **instrumentation amplifier,** which amplifies the difference between the electrodes and rejects the common mode signals. Next, the signal goes through a **low-pass filter** stage to reduce noise, and amplify the signal. A 60Hz **notch filter** is then implemented to eliminate all grid noise. After the 60Hz is filtered out, the signal goes through another **low-pass filter stage.** This low-pass filter stage has the same cut-off frequency, but has a variable gain based on the user's preference. Before the summing amplifier, the signal is put through a **buffer** in order to isolate the stages. The **summing amplifier** raises the signal voltage in order to make the whole signal positive. The signal is then input into the **MSP430 microcontroller**, which implements a REM algorithm to determine when the user is in REM sleep. The microcontroller outputs a low frequency to the piezoelectric **speaker** when the user enters REM sleep, and a high frequency when the user exits REM sleep. The microcontroller also turns on indication **LEDs** to help the user set their gain. Additionally, the MSP430 converts the signal from analog to digital and outputs that digital signal to the **Bluetooth Module**, which then transmits the signal to a pc. This allows the user to view and log their sleep data.

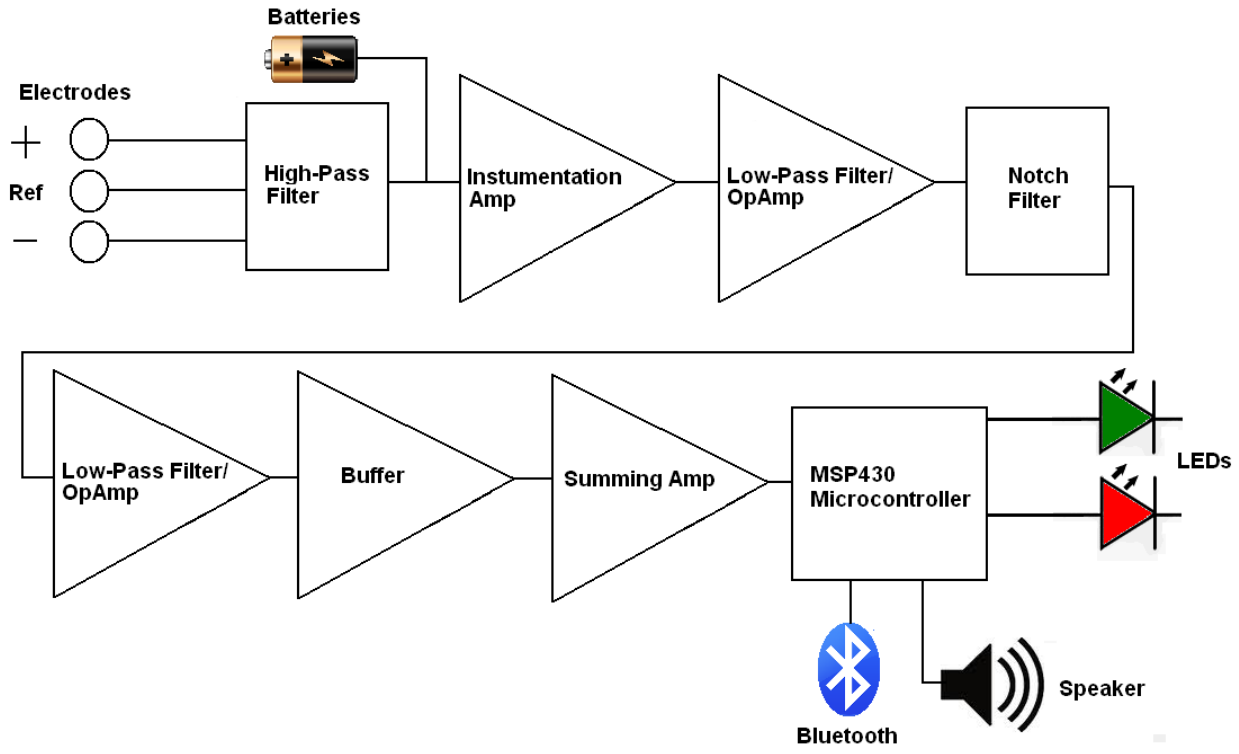


Figure 3: Block Diagram of Circuit

Description of Blocks:

Below is a detailed description of the schematic, including the algorithms used to implement the design. Refer to each block in schematic Figure 3.

Battery: This device is completely battery operated to ensure user safety. A total of four batteries are used to create +3V, -3V, and GND. All active devices are supplied with +/-3V in order to capture the positive and negative cycles of the EOG. Capacitors C11, C12, C13, and C14 are used as bypass capacitors intended to decrease noise from the power source.

Electrodes: There are a total of three electrodes. The (+) and (-) electrodes are placed above the eye, and the reference electrode is placed between the eyes. The electrodes detect the polarity across the eye as the eye moves.

High-Pass: There is a passive high-pass filter off of the (+) and (-) electrodes. The high pass filter has a cut-off frequency of 0.16Hz, and blocks any DC offsets.

$$f_c = \frac{1}{2\pi R_1 C_1} = 0.16Hz$$

Instrumentation Amp: The INA126 instrumentation amplifier is used in this device. The INA126 has a CMRR of 94dB, which is higher than the typical instrumentation amp. A high CMRR is needed in order to reject the common mode signals from the human body. This instrumentation is intended for physiological signals such as the EOG, ECG, EEG, and EMG. The gain of this stage is 123.5, this was chosen because increasing the gain further could decrease the CMRR. There is a passive high-pass filter following the instrumentation amp with a cut-off frequency of 0.8Hz to block DC drifts.

$$G = 5 + \frac{80k\Omega}{R_3} = 123.5$$

$$f_c = \frac{1}{2\pi R_4 C_3} = 0.8Hz$$

Low-Pass/Gain: The MCP602 is used to create active filter stages. The first low-pass filter has a cut-off frequency of 34.5Hz and gain of 10. A cut-off frequency of 34.5Hz is used because the resulting cut-off frequency of the cascaded filters is ≈10Hz.

$$G = 1 + \frac{R_5}{R_6} = 10.1$$

$$f_c = \frac{1}{2\pi R_5 C_4} = 34.5Hz$$

Notch Filter: The Twin-T notch filter was used in this stage. This filter has a cut-off frequency of 60Hz to eliminate the grid noise.

$$f_c = \frac{1}{2\pi R_{10} C_6} = 60.01Hz$$

Low-Pass/Gain: The second low-pass filter stage also has a cut-off frequency of 34.5Hz. However, this stage has a potentiometer so the user is able to adjust the gain. The nominal gain is 10, but the user can increase or decrease the potentiometer depending on their internal resistance. The potentiometer has a range from 0-25kΩ over 10 turns which gives the user increased accuracy.

$$G = 1 + \frac{R_{11}}{R_{pot}} = Variable$$

$$f_c = \frac{1}{2\pi R_{11} C_9} = 34.5Hz$$

Buffer: After the second Low-Pass filter stage there is a voltage divider to reduce the signal voltage in half. A buffer is then used to isolate the previous stage from the next stage, and prevent loading affects.

$$V_{out} = V_{in}\left(\frac{R_{15}}{R_{15} + R_{14}}\right) = \frac{1}{2}V_{in}$$

Summer: A summer circuit is used to raise the voltage by +1.5V. In order to use the standard ADC pins on the MSP430 microcontroller the inputs must be positive. The summing circuit raises the signal voltage from -1.5-1.5V to 0-3V.

$$V_{out} = -V_1\left(\frac{R_{18}}{R_{17}}\right) - V_2\left(\frac{R_{18}}{R_{18}}\right) = -(V_{in} + 1.5V)$$

Inverter: Since the summing circuit inverts the signal, we added an inverting amp with a gain of 1 to invert the signal back.

MSP430: The MSP430 microcontroller is used to detect REM and alert the user. See the Software section for REM detection algorithm details. The microcontroller is powered by +3V and GND. The REM signal inputs to pin P1.1. The RST pin is connect to the RESET switch. The MSP430 outputs PWM to P1.6 to the speaker. Pin P1.2 outputs to the RXD input of the Bluetooth Module. Pins P1.0 and P1.4 output to the indication LEDs.

REM Indication lights: There are two indication LEDS on the PCB. The Red LED (P1.4) indicates that the user's eye movements are not in the REM threshold. The Green LED (P1.0) indicates that the user's eye movements are in the REM threshold. One LED will always be illuminated.

Reset Switch: The RESET switch allows the user to reset the microcontroller. The RESET pin is pulled high to +3V, and is activated when grounded. When the RESET pin is activated it allows the user to reset device without having to pull power to the whole circuit.

Speaker: The speaker has 2 modes: low frequency (250Hz) mode, high frequency (2500Hz) mode. The low frequency mode is sounded when the user enters REM. This soft noise allows the user to recognize the sound while dreaming in the attempts to lucid dream. When the user exits REM the harsh high frequency noise will sound to wake the user to help remember the dream experience.

Bluetooth: The Bluetooth module transmits the signal data to a pc with internal Bluetooth or a USB Bluetooth receiver. From there the user can view and log their sleep data. (See LabVIEW section for more details)
The pin assignment is as follows:
VCC: +3V
GND: GND
TXD: Open
RXD: P1.2 of MSP430

Figure 4: JY-MCU Bluetooth Module

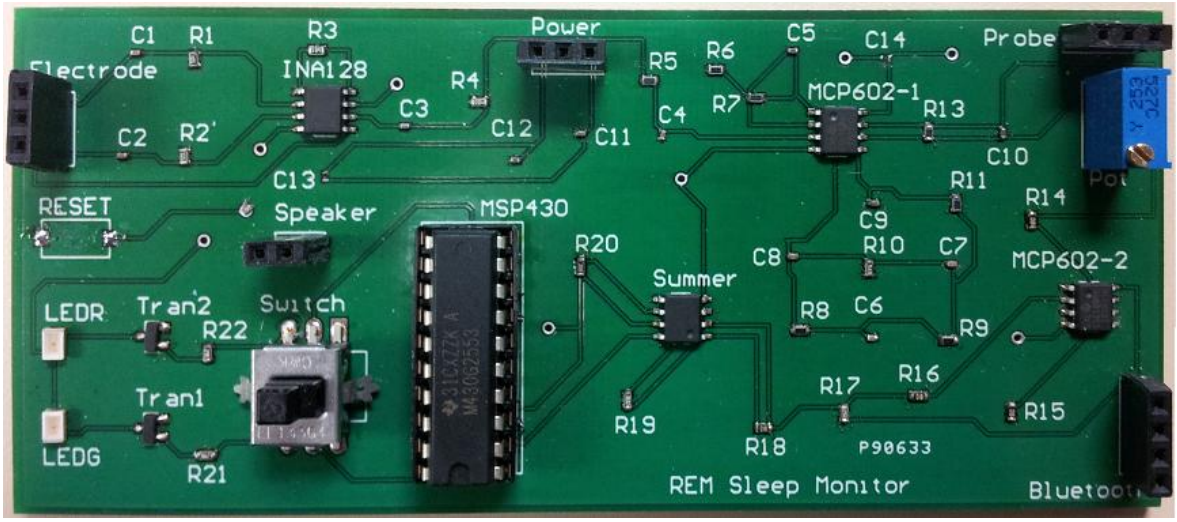


Figure 5: Printed Circuit Board (PCB)

Schematic:

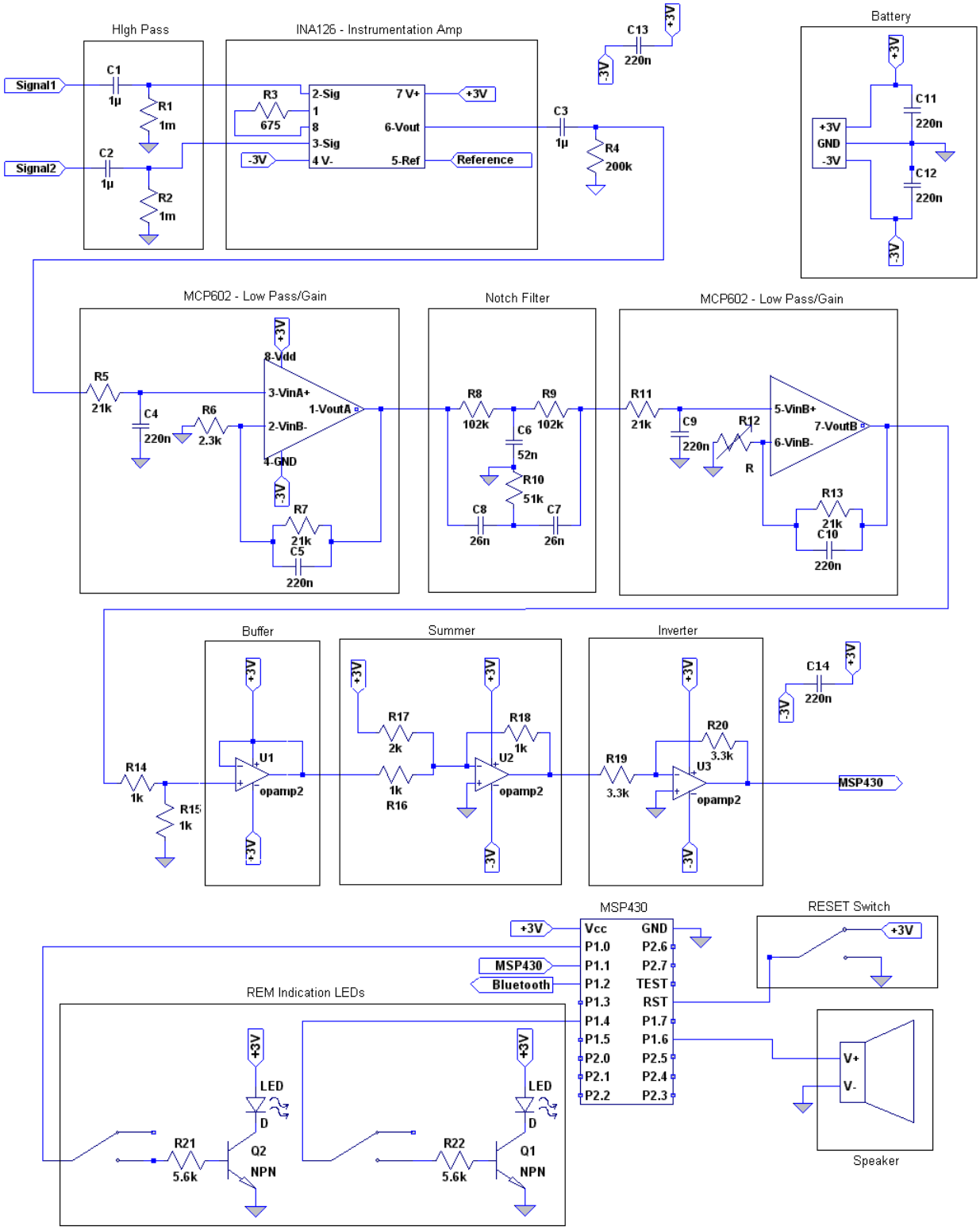


Figure 6: Circuit Schematic

Filter Design:

In order to acquire the cleanest signal, the EOG signal (≈1-10Hz) is fed through several filter stages.

High-Pass

A high-pass filter with cut-off frequency of 0.16 is implemented before the instrumentation amplifier in order to block DC offsets. The signal goes through another high-pass filter, with cut-off frequency of 0.8Hz, after the instrumentation amplifier since the EOG signal ranges from 1-10Hz.

Low-Pass

A cut-off frequency of 34.5Hz is used for the two low-pass filter stages because the resulting cut-off frequency of the cascaded filters is ≈10Hz.

Notch

A notch filter at 60Hz is implemented in between the low-pass stages to eliminate all grid noise. The notch filter was placed between the low-pass stages for testing and troubleshooting purposes.

Below is the frequency response of the low-pass(blue), high-pass(green), and notch(cyan) filter:

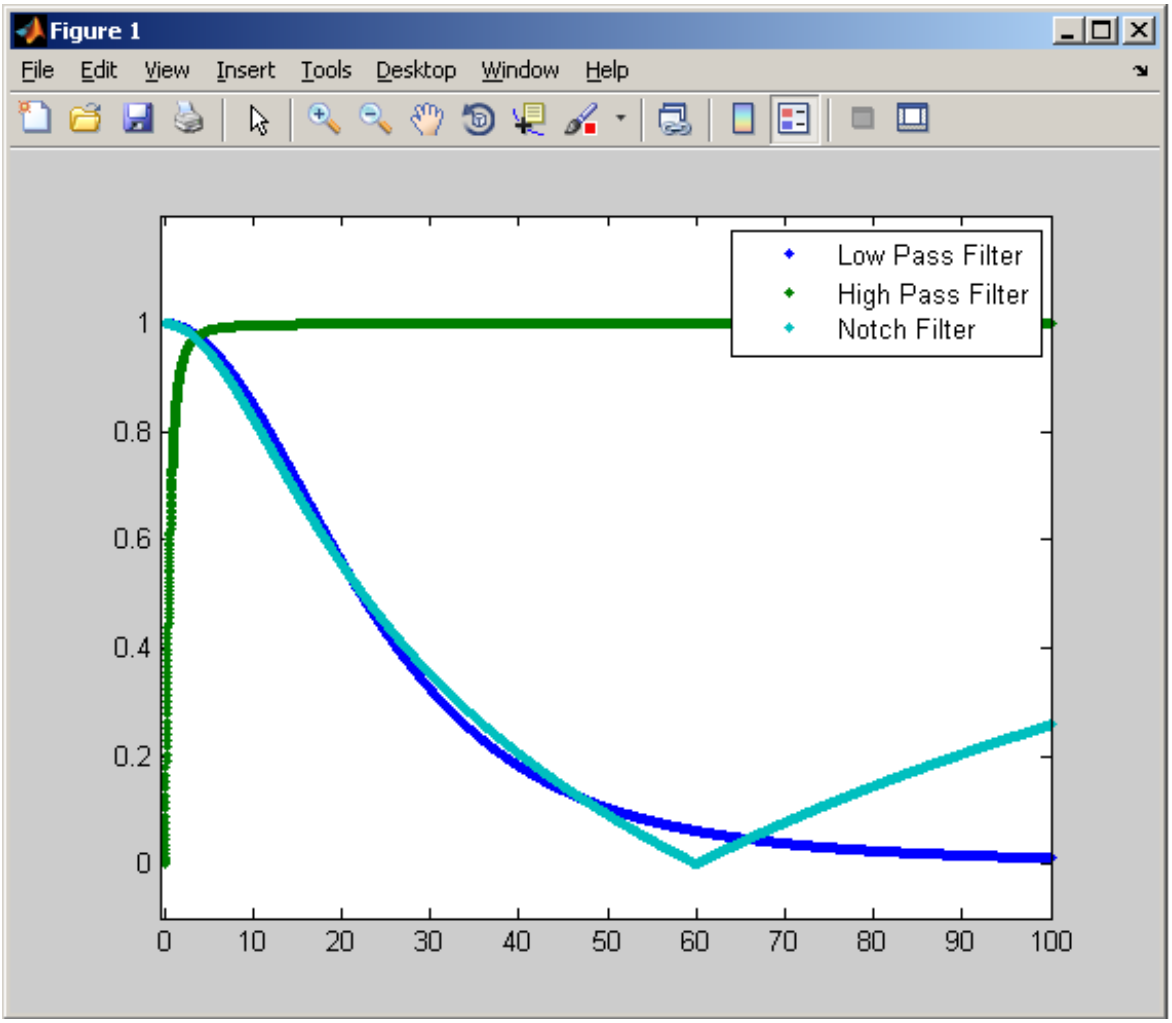


Figure 7: Frequency Response of Low-Pass, High-Pass, and Notch Filter

The filter stages cascaded together result in a band-pass filter from ≈1-10Hz, which captures the full frequency range of the EOG signal

Below is the frequency response of the final filter with all of the stages cascaded:

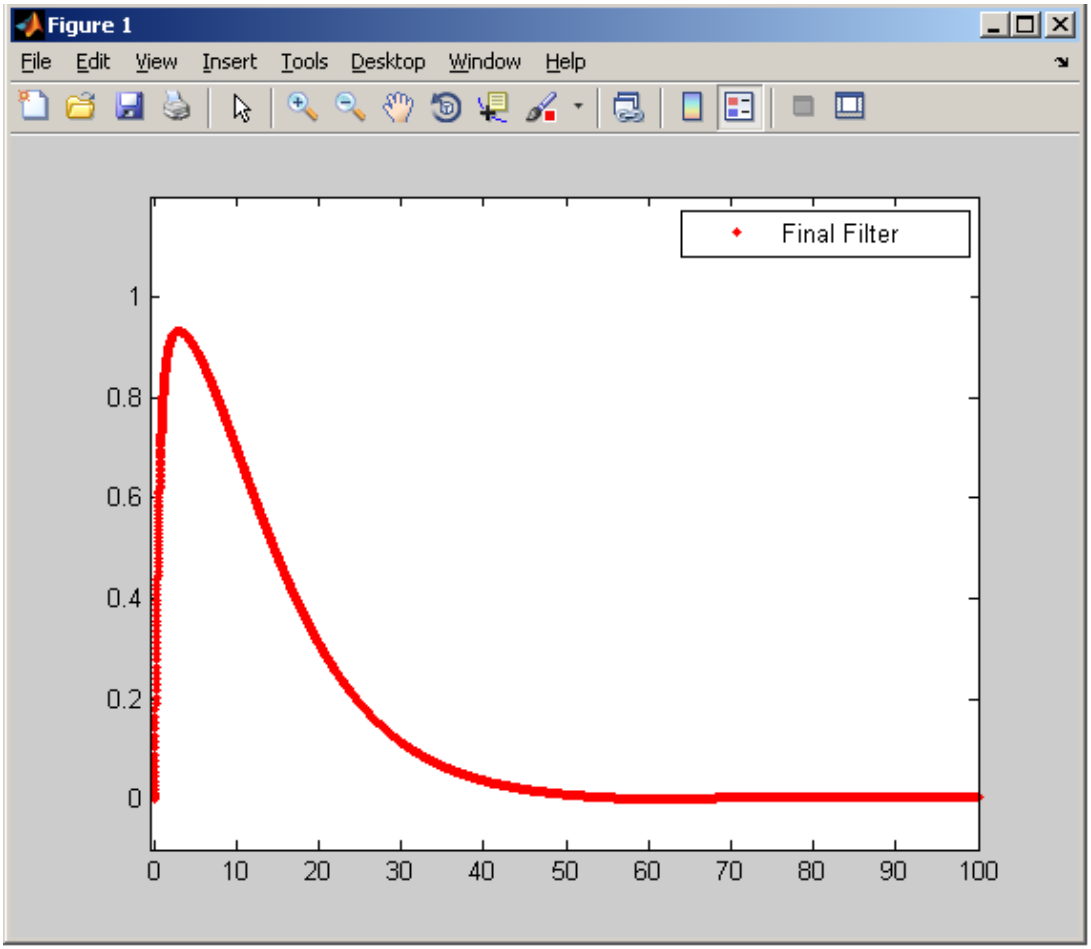


Figure 8: Total Frequency Response of Filters

Software

LabVIEW:

This device utilizes LabVIEW to graph and log sleep data. The Bluetooth module transmits the signal data to a pc with internal Bluetooth or a USB Bluetooth receiver. LabVIEW then converts the data from hexadecimal to decimal and graphs the signal. This allows the user to view their EOG without the use of an oscilloscope. LabVIEW displays a real time graph of the user's EOG, along with the start and end time of the sleep period. Data automatically saves to a file every time the REM.vi is started.

Front Panel

When the user opens the REM.vi in LabVIEW the front panel will appear. The front panel is the user interface which allows the user to run and stop the vi. There, the user can choose which COM Port they are using. Below, the front panel is displayed:

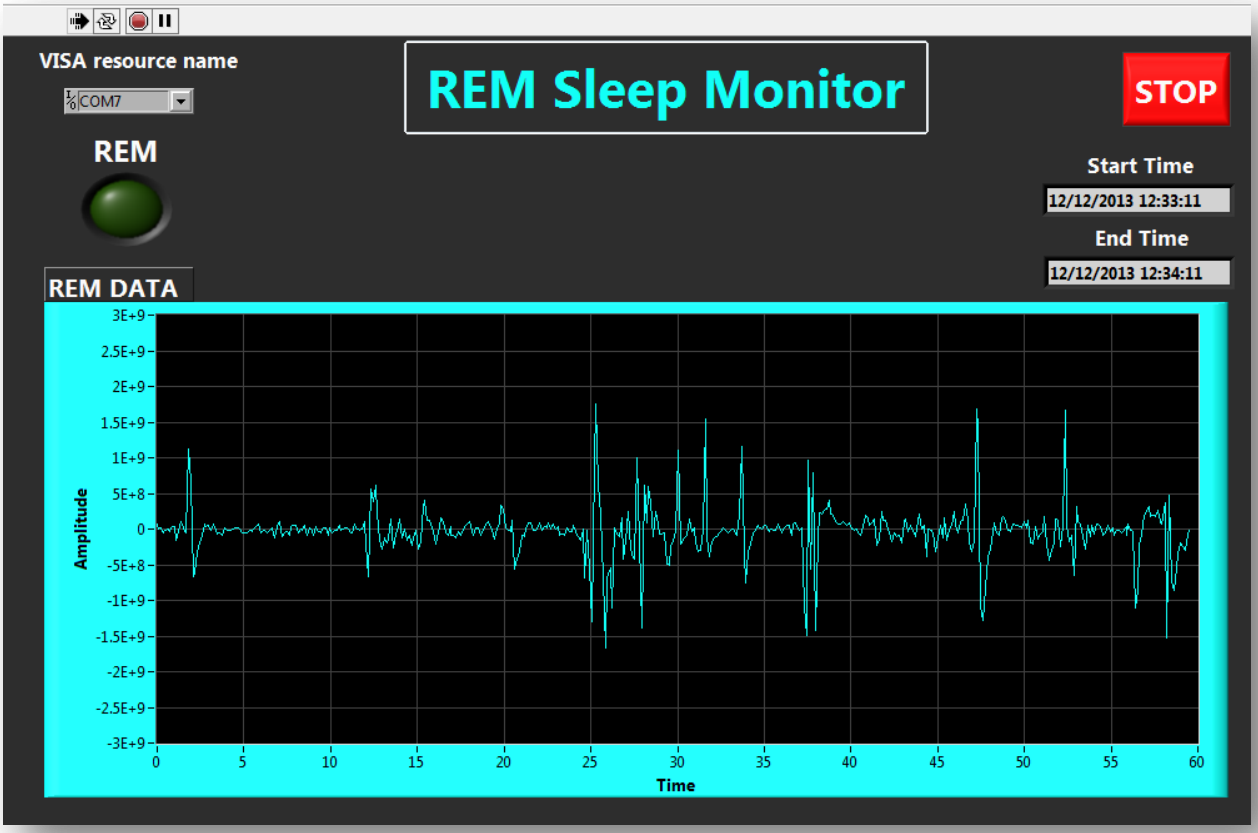


Figure 9: Front Panel

Block Diagram

The block diagram is where the program is created, and allows the user to edit the program. Here, the user can edit where the data file is saved, along with other settings. First the program opens and initializes the user controlled COM Port. Then the program reads the data received by the COM Port and converts it from hexadecimal to decimal. From there LabVIEW saves the data to a .lvm file and graphs the data. The Elapsed Time vi displays current time, as well as keeps track of the elapsed time. Below, the Block Diagram is displayed:

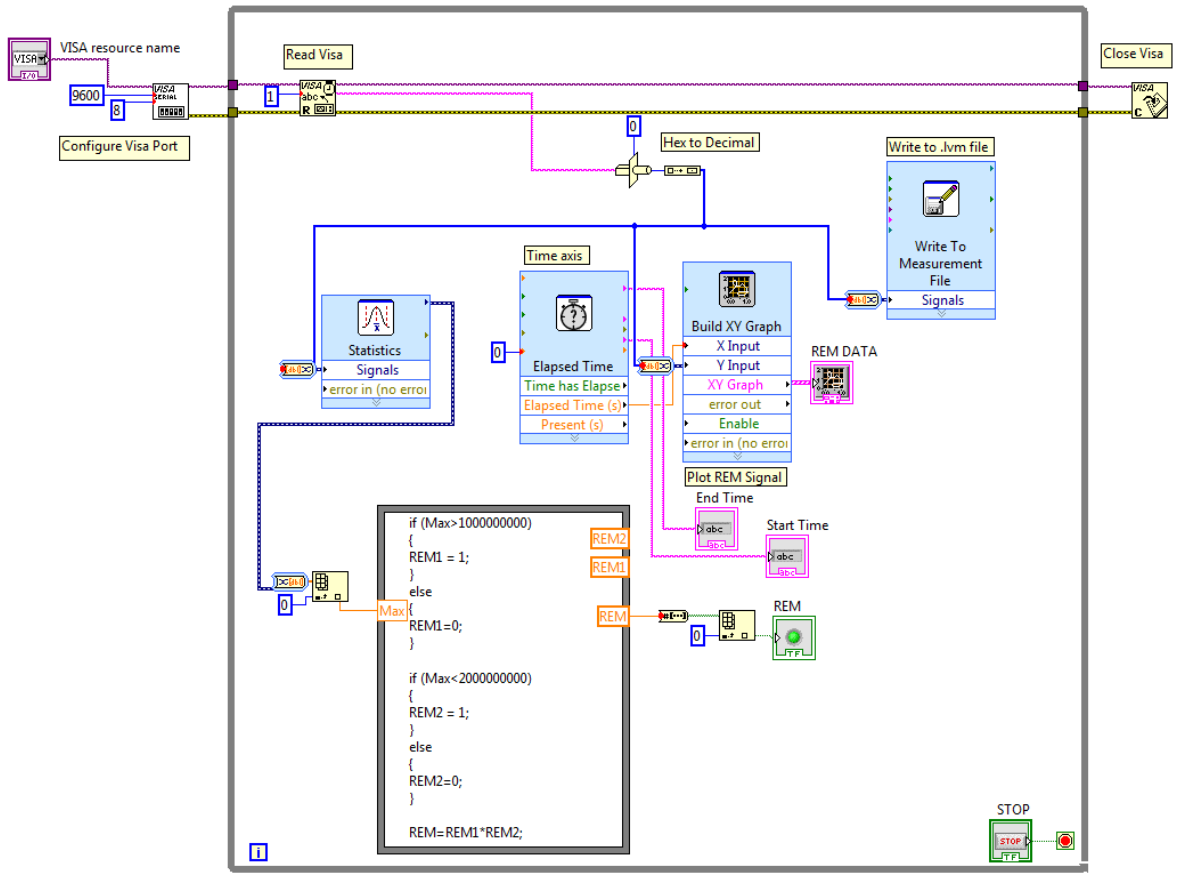
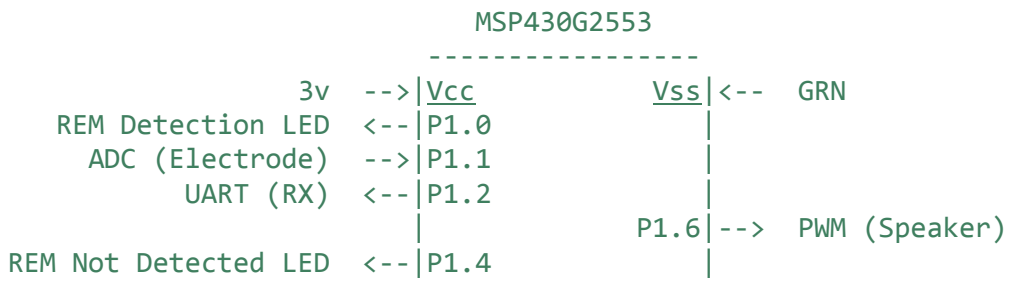


Figure 10: Front Panel

MSP430 Microprocessor:

This section will go into detail about some features a user can change by programming the MSP430G2553.



Initialize Variables

Initializing the following variables will allow the user to set up the REM Monitor to their own personal ideal settings.

- `int ScanPeriodInREM = 457;  // 457 is approx 30 seconds`

This variable will allow the user to program a scan period for REM. The scan period acts as a checkpoint, if REM is positive during this scan period the checkpoint is successful. Note when setting ScanPeriodInREM, if the period is too long then detecting REM may not be accurate. For example if ScanPeriodInREM is 5 minutes, it is possible that the user was in REM for the first 5 seconds of the 5 minute scan period, this would result in a successfully checkpoint even though the individual may have come out of REM sleep. Another example, if ScanPeriodInREM is 0.2 seconds, the user could be in REM but in order to turn on the Low Frequency speaker a desired NumberOfScanPeriods needs to be met in a row. Choosing 30 seconds maybe a good medium.

To solve for ScanPeriodInREM:

$$\text{"Seconds in ScanPeriodInREM"} = (ScanPeriodInREM) * 0.065536$$

Note that 0.065536 comes from the clock settings and how many bits the MSP430G3553 is. The MSP430G2553 is 16 bits and the clock is set for 1Mhz.

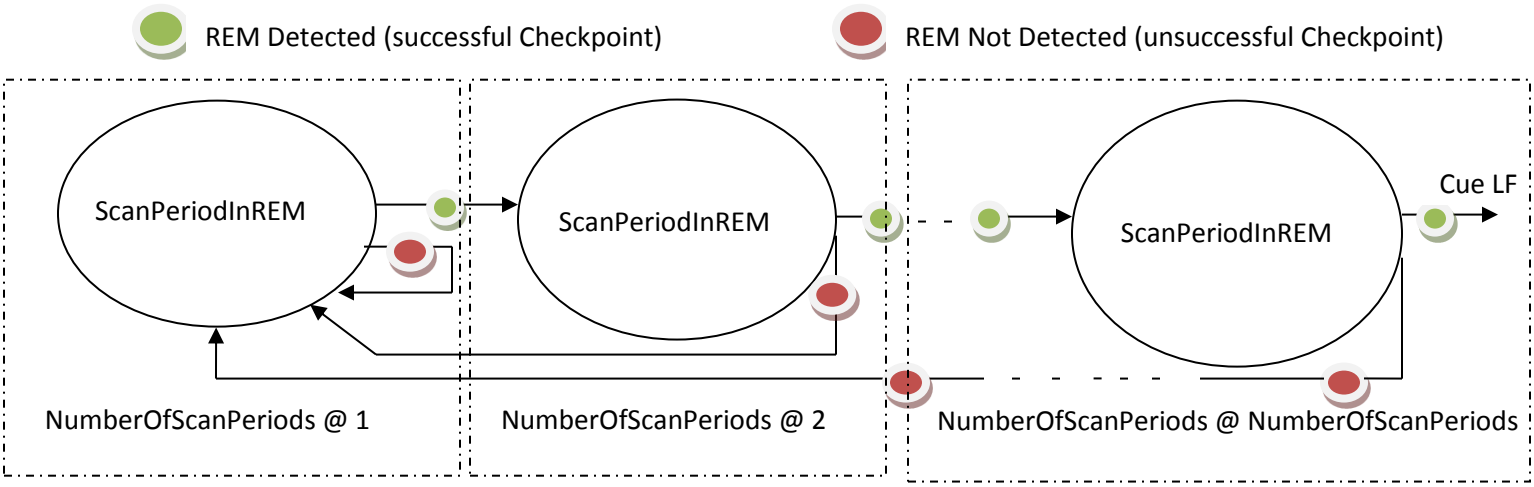
$$\text{"Time per Clock cycle"} = \frac{2^{16}}{1\text{Mhz}} = \frac{65536}{1000000} = 0.065536$$

- `int NumberOfScanPeriods = 10;  // 10 periods is approx 5 mins total time scanning`

This variable will decide how many checkpoints (ScanPeriodInREM) are needed to cue the user if they are lucid dreaming. Setting this variable will choose how long an individual will be in REM before cueing; this variable is affected by ScanPeriodInREM as shown below.

$$\text{"Time in REM"} = (NumberOfScanPeriods) * (ScanPeriodInREM)$$

In the code set up ScanPeriodInREM = 457 and NumberOfScanPeriods = 10. So this means that approximately 300 seconds are needed in REM before the speaker is turned on at a low frequency. Note that if a checkpoint is unsuccessful then NumberOfScanPeriods will go back to zero. If enough data is gathered for the individual, that data could be used to set ScanPeriodInREM and NumberOfScanPeriods which would increase the accuracy of detecting REM.



- `int MinsInLowFre = 5;  // 5 is approx 5 mins that Low Frequency speaker is sounded (this is the lucid dream cue)`

This variable will determine how long the speaker is sounded at a low frequency. The number that MinsInLowFre equals will be how many minutes that the speaker is sounding at low frequency before it sounds in high frequency. Low frequency is used to cue the user that they are dreaming. Note in the code example, REM was scanned for 5 minutes then low frequency was sounded for 5 minutes, making a total of 10 minutes that an individual should be in REM. The goal is to have the speaker sound at a high frequency to wake the user directly after REM. With this goal, the user would be able to remember the lucid dream experience.

Registers

The following statements are used to set up the MSP430G2553. All of the registers can be found in the MSP430G2553 data sheet. The following section will going into more detail about the more complicated registers.

- `TACTL = TASSEL_2 + MC_2;  // Set the timer A to SMCLCK, Continuous`

This statement sets up the clock at 1Mhz, changing this would result in different times for all statements. Note that this statement is used later on to change the clock speed for pulse wave modulating.

- `ADC10CTL1 = INCH_1; // input A1`

This statement sets up the ADC (Analog to Digital Converter) reference voltages. INCH_1 uses the battery as reference voltage, making whatever voltage is on VCC (up to 3.6v) is the high reference and whatever voltage is on GND (down to -2 volts) is the lower reference. Note that the maximum potential difference between VCC and GND can only be 4volts. The MSP430G2553 has internal references that can be used, one at 2.5v, but we chose to use the battery as a reference voltage to give us a wider range. Changing this will affect the ADC values.

Main

Below is the main code, this section will go more into detail about the algorithm.

- `_BIS_SR(LPM0_bits + GIE);  // Enter LPM3 // Exit is located in interrupt, not efficient with no crystal and ADC interrupt`

The statement above enters the MSP430G2553 into low power mode. The goal of this is to save battery life while the user is still awake and attempting to fall asleep. Low power mode can be exited after a chosen period of time found in the timer interrupt service routine. This is a feature that can be implemented if desired. LPM is not used for now for a few reasons: using the LPM3 without a crystal is said to be inefficient, the interrupt routines may make LPM inefficient. Until LPM is proven to save energy in our circuit it will be left out.

- `if (ADC10MEM < 0x266)  //Low REM Voltage (1.8V is where REM starts, REM is between 1.8 and 2.8V)`

The content of this "if" statement is the algorithm for detecting REM. ADC10MEM stores the digital value from the electrodes that the user is wearing (analog signal has been converted to digital, ADC10MEM is the digital value for the analog signal) and then ADC10MEM is compared to a chosen threshold (value). This value is set for a lower threshold, everything below this value in not considered as finding REM. This threshold is used to differentiate between reference noise and actual eye movements. The main goal of this threshold is to not include noise from the power grid or the circuit to be tracked as REM. To solve for the lower threshold see below:

$$\frac{\text{Desired threshold}}{\text{ADC Register Reference}} * (\text{ADC \# bits}) = \text{"Decimal value for ADC threshold"}$$

For an example I will use 1.8v as a desired threshold.
The ADC Register Reference is selected to use our battery as the reference voltage (as seen above in the register section), our battery is 3v to 0 (GND).
The MSP430G2553 has a 10 bit ADC

$$\frac{1.8\text{v}}{3\text{v}} * (2^{10}) = \text{"Decimal value for ADC threshold"}$$

$$0.6 * 1024 = \text{"Decimal value for ADC threshold"}$$

$$614.4 = \text{"Decimal value for ADC threshold"}$$

After this the decimal value 614 was converted to a hexadecimal value which equals 266. If ADC10MEM is less than 1.8 volts it will enter this "if" statement. During this statement an LED indicator is triggered to notify the user that they are not in REM.

```
{
        P1OUT &= ~0x01;                 //Clear P1.0 LED off
        P1OUT |= 0x10;                  //Set P1.4 LED on
}
```

A lower threshold at 1.8v was chosen because our signal is centered at 1.5v, setting 1.8 as the lower threshold helped cancel out any other noise that was not taken care of by our hardware filters.

❖ `else if (ADC10MEM > 0x3BB)  // High (over 2.8V is saturated)`

The next part of the "if" statement is the "else if" for the upper threshold. This is used to cancel out any eyebrow movements or other muscle movements not from the eyes. If large muscle movements occur the signal will saturate at 3 volts, these movements should not be counted as REM. To pick a proper threshold it is important to see the maximum voltage that REM reaches. After the max REM voltage is found everything above should not be considered as eye movements, so the high threshold can be chosen to be at or a little above the maximum REM voltage, this will differ between people (note that our hardware has a gain control to adjust the gain of the signal). We found that 2.8v was a proper high threshold for our testing. Using the method above, a chosen voltage at 2.8 makes a hexadecimal value of 3BB.

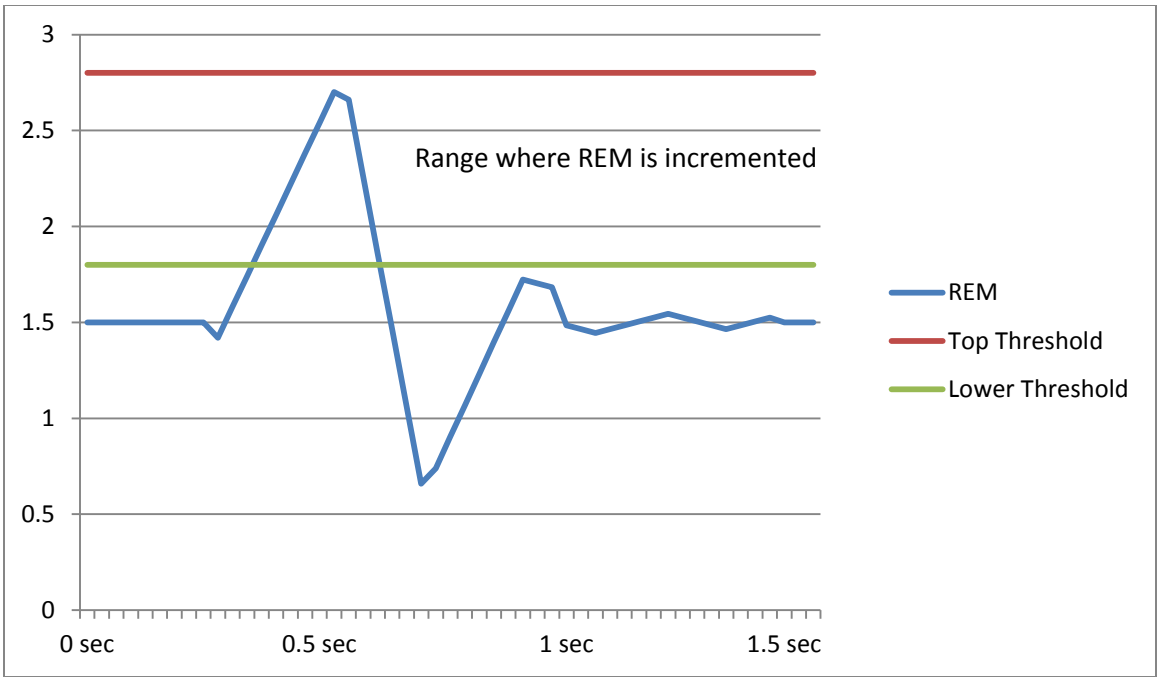
```
{
        P1OUT |= 0x10;      //Set P1.4 LED on
        P1OUT &= ~0x01;     //Clear P1.0 LED off
        REM--;              // Subtracts to REM if in voltage range
}
```

When the "if else" statement is entered, REM is decreased for the amount of time the signal is over the chosen high threshold. Anything inside the threshold increments the 'REM' variable. Anything over the upper threshold decrements the 'REM' variable. This is done to eliminate the REM detection during eye movements or large muscle movements. Also the "REM detection" LED is off and the "REM not detected" LED is on.

❖ `Else`

The last part of the REM detection code is an "else" statement that increments REM. In the above scenario, for signals below the lower threshold or when the signals are greater than the upper threshold, REM is not detected. All that is left is the scenario when the signal is within the upper and lower thresholds. In this case REM is detected and the "REM detection" LED is on and the "REM not detected" LED is off.

```
P1OUT |= 0x01;      //Set P1.0 LED on
P1OUT &= ~0x10;     //Clear P1.4 LED off
REM++;              // Adds to REM if in voltage range
```



What is expected when an eye movement is picked up by the REM Monitor II, Voltage with respect to time in seconds

Figure 11: REM Threshold

The "if" statements seen between the "REM increment" and PWM code are directly related to the set variables in the "Initialize Variables" section. Look at the above "Initialize Variables" to understand the algorithm between the REM detection algorithm and the Pulse Wave Modulation code.

Pulse Wave Modulation

Once the desired "NumberOfScanPeriods" is met, the "timercount" variable will be greater than 460, this will cause the code to stay out of all "if" statements related to "ScanPeriodInREM" and "NumberOfScanPeriods". ADC values are still taken in and sent via UART to the Bluetooth, and the REM indicator lights are still active. When the code enters the PWM stage, it will eventually require a hard reset.

```
❖ if((timerCount>465))      // For this statement timer is sped up, 1second = 500
  timerCount
          {
                P1DIR |= (1<<6);              //P1.6 output
                P1SEL |= (1<<6);              //P1.6 TA1/2 options
                CCTL1 = OUTMOD_6;           // CCR! toggle/set
                TACTL = TASSEL_2 + MC_3;   // SMCLK, up-down mode
```

P1.6 is used as an output for the speaker; the above lines initialize P1.6 for the speaker. CCTL1 then sets up P1.6 so that it can toggle to PWM and TACTL sets the clock, more details about this can be found in the MSP430 data sheet.

```
CCR0 = 1000;                    // PWM Period/2
```

$$\left(\frac{\text{CCR0 Value}}{4*\text{Clock Speed}}\right)^{-1} = \text{Frequency}$$

$$\left(\frac{1000}{4*1000000}\right)^{-1} = \text{Frequency}$$

$$(0.004)^{-1} = \text{Frequency}$$

$$250\text{ hz} = \text{Frequency}$$

The frequency can be controlled by setting CCR0. The CCR0 value is a multiple the period, therefore you must divide it by 4. This equation will work for picking a frequency.

The next change that can be made is with the duty cycle.

```
CCR1 = 5;                              // CCR1 PWM duty cycle
```

The duty cycle is easy to set, all that is required a percent on time, and that value can be set for CCR1.

$$\frac{\text{CCR1}}{100} = \text{Duty Cycle}$$

$$\frac{5}{100} = \text{Duty Cycle}$$

$$5\% = \text{Duty Cycle}$$

For this example the duty cycle is 5%, this means that the wave will be high 5% and low 95%, in other words, for five clock ticks the wave will be high and for 95 clock ticks the wave will be low. This can be used to adjust the volume for each individual user. We have set our duty cycle low because we do not want the user to be awakened when being cued while they are in REM.

The high frequency PWM is basically the same. The two changes are to the frequency and the duty cycle.

```
CCR0 = 100;             // PWM Period/2
CCR1 = 50;              // CCR1 PWM duty cycle
```

The same equations can be used as above to solve for these values, the frequency comes to being 2.5 Khz and the duty cycle is 50%. We chose a high frequency with a higher duty cycle to make sure the user would wake up from this tone.

Interrupts

The follow section will discuss the interrupts that are used and explain in a little more detail what is happening in the interrupts.

```
// Timer A0 interrupt service routine
❖ #pragma vector=TIMER0_A0_VECTOR
  __interrupt void Timer_A (void)
```

Setting up the TIMER0 interrupt can be found in the MSP430G2553 data sheet, the main thing to look at is what's happening inside this interrupt. There first thing that happens in this interrupt is timerCount is incremented

```
❖ timerCount = (timerCount + 1); // Clock at 1Mh (time per clock =65536/1000000)
```

The timerCount variable is used to keep track of time in an easy straight forward manner. We solved for how much timerCount equals earlier, by dividing the amount of data by the clock. Every timerCount value stands for a certain amount of time seen below.

$$(\text{timerCount}) * (0.065536) = \text{"Time in Seconds for timerCount"}$$

For example if timer count equals 457 how many seconds would timerCount be.

$$(457) * (0.065536) = \text{"Time in Seconds for timerCount"}$$

$$29.95 = \text{"Time in Seconds for timerCount"}$$

This is how the time periods are set to a chosen value, making the program very usable and easy to change time periods. The next statement in the TIMER0 interrupt is related to the ADC data being sent.

```
❖ if(timerCount%2)        // Send data every other timerCount, (too much data slows
  LabVIEW)
```

During our testing process we tried collecting and sending data in different places of the code and at different speeds. If the data was sent too fast LabVIEW (the program that was processing the data) would not show the data real time. We found that sending the data every other timerCount (about every .13 seconds) would keep LabVIEW running real time. The modulus is set to modulate timerCount by 2 making the "if" statement only entered on every other timerCount increment. Note that it is possible to send data faster, but for our demonstration purposes this was enough. When looking at an 8 hour period of data collection, this comes to be a large quantity of data even with the slower send rate.

```
❖ UCA0TXBUF = ADC10MEM;  // UART stored and sent in UCA0TXBUF
```

The only statement inside the "if" statement is the UART store and send line. ADC10MEM stores the ADC value and UCA0TXBUF transmits the data. For more information on this refer to the MSP430G2553 data sheet. The next statement is related to the Low Power Mode.

```
❖ if(timerCount>18280) // 18280 is approx 20 min, MSP430 is in sleep mode for 20
  min // Used for Low power mode, With ADC, interrupt, and no crystal Not Efficient
      {
            _BIC_SR_IRQ(LPM0_bits);                  // Clear LPM3 bits from 0(SR)
      }
```

As mentioned above at the start of the Main, the Low Power mode is an option. This is the exit service routine. The wait period can be set to whatever desired amount by setting the value in the "if" statement.

```
❖ // ADC10 interrupt service routine
  #pragma vector=ADC10_VECTOR
  __interrupt void ADC10_ISR(void)
     {
     __bic_SR_register_on_exit(CPUOFF);            // Clear CPUOFF bit from 0 (SR)
     // __delay_cycles(500); If adc acts to fast, delay can be used
     }
```

For the last part of the code the ADC10 interrupt is enabled. This sets up the ADC and makes and stores the digital value for the analog point in ADC10MEM. The only other line not strictly related to the ADC is the __delay_cycles(500) this is used to slow the ADC down in case it reads or transmits to fast. This problem is solved in the above timer interrupt. For more information about the ADC10 interrupt service routine refer to the MSP430G2553 data sheet.

Code Summary

The last few pages described in detail some of the more complicated parts of the code. The software used for programming the MSP430 is Code Composer Studio 5.1. The software is free and there are many tutorials available online for starting a MSP430 project.

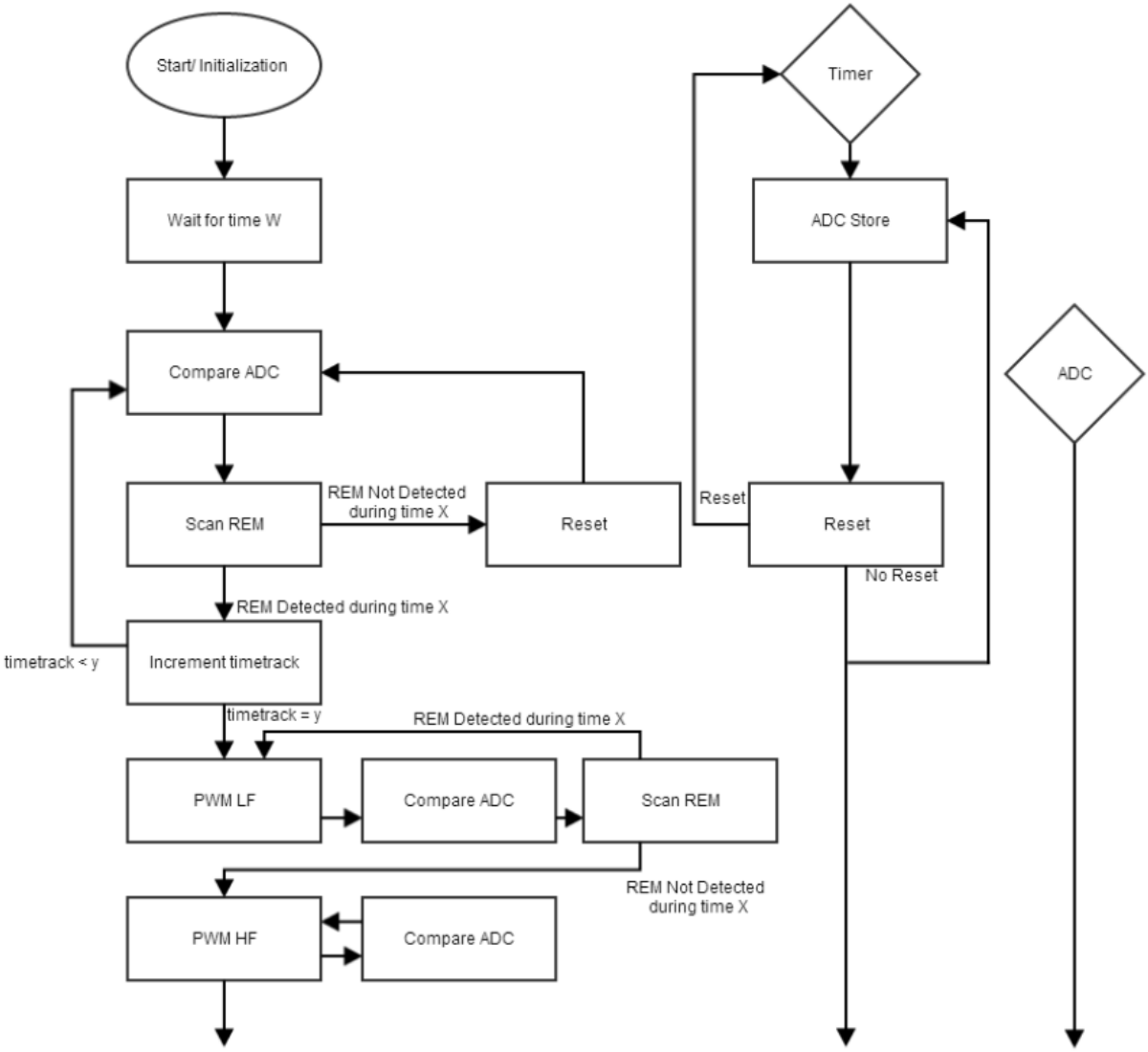


Figure 12: Software Flow Chart

Troubleshooting

PCB:

The PCB has two probe point headers built in for troubleshooting. The probe points are located at the top right corner of the PCB.

P2

Probe point P2 reads the signal before the voltage divider and summing circuit. This EOG signal should range from -3V to +3V. If this signal is not what is expected, then the user should check the power connection and electrode connections. If both of those connections are solid, then there is trouble with the INA instrumentation amplifier, or the filter stages.

P1

Probe point P1 reads the signal right before it is fed into the microprocessor, this EOG signal should range from 0V to +3V. If P2 is functioning normally and P1 is not, then there is a problem in the summing circuit or the inverter.

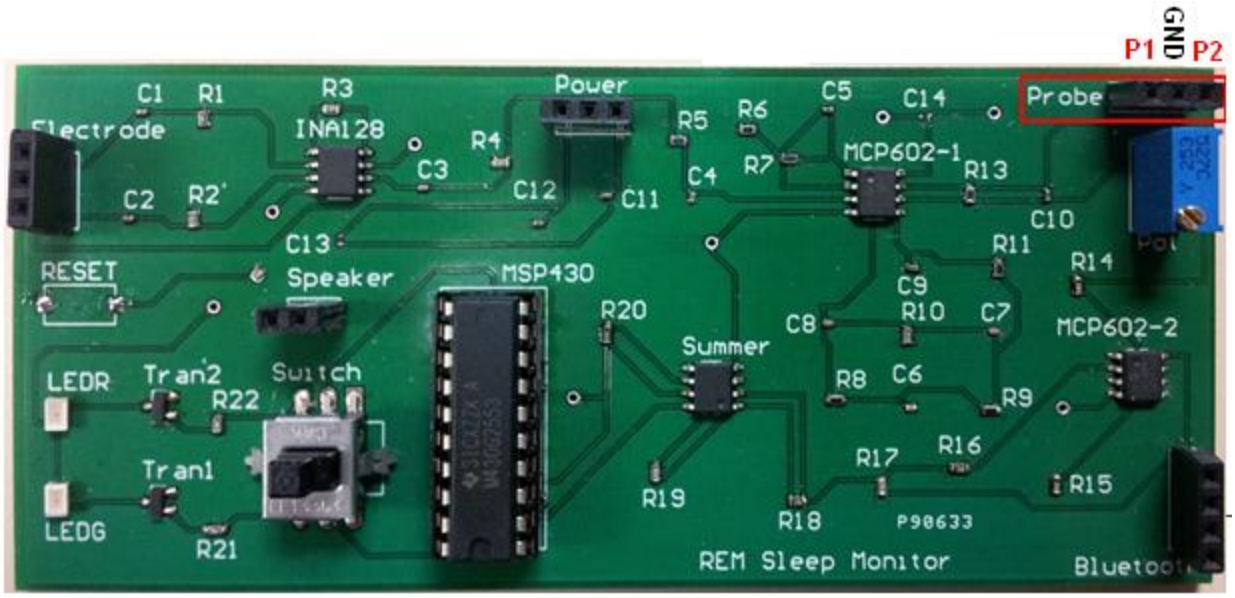


Figure 13: Probe Header PCB

Proto-Board:

There are endless places to troubleshoot on the proto-board.

P2

Probe point P2 reads the signal before the voltage divider and summing circuit. This EOG signal should range from -3V to +3V. If this signal is not what is expected, then the user should check the power connection, and electrode connections. If both of those connections are solid, then there is trouble with the INA instrumentation amplifier, or the filter stages.

P1

Probe point P1 reads the signal right before it is fed into the microprocessor, this EOG signal should range from 0V to +3V. If P2 is functioning normally and P1 is not, then there is a problem in the summing circuit or the inverter.

P3

The user can check the filter stages by applying a sine wave to probe point P3, and reading the output and point P2. The user can vary the frequency of the sine wave and see if the filters are collectively acting as a 1-10Hz band-pass filter. From there the user can test each filter individually if need be. (Refer to the How To Guide for more details about the proto-board)

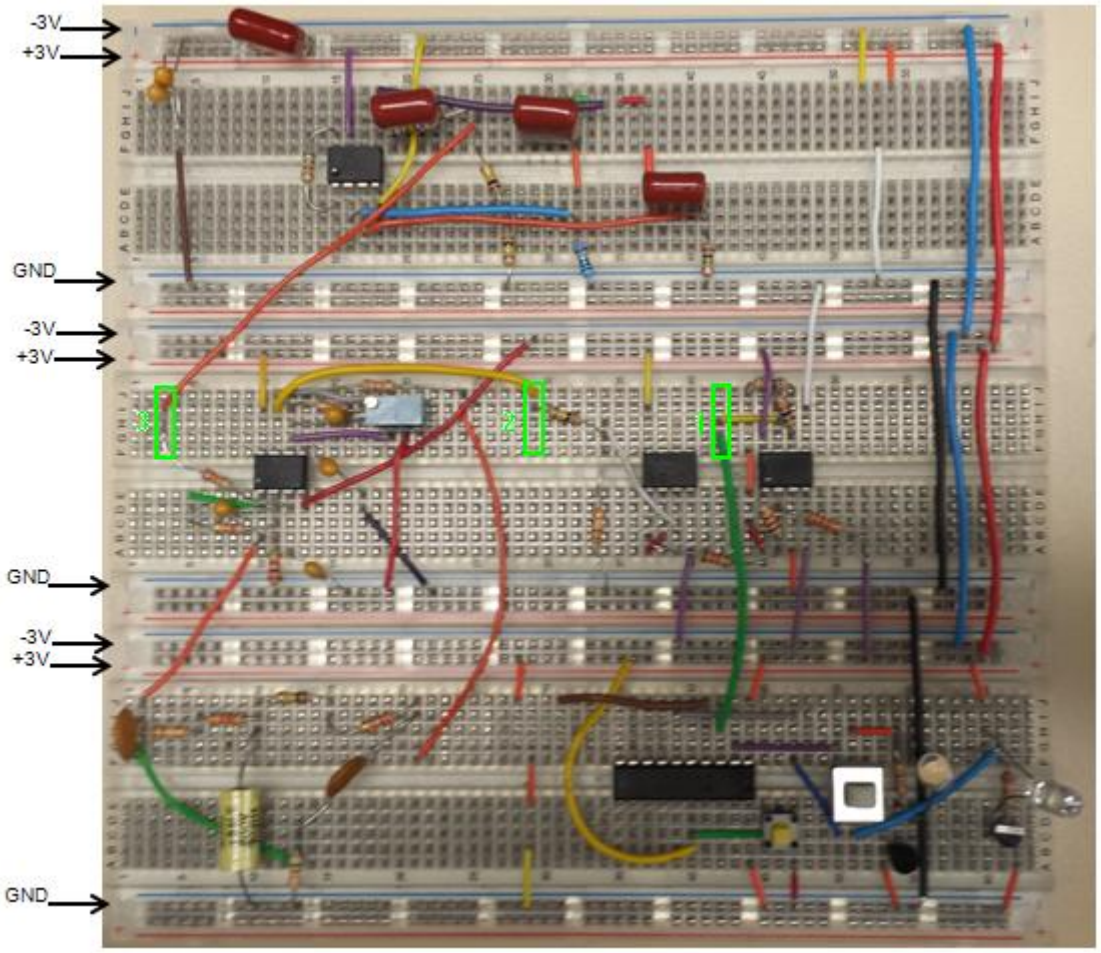


Figure 14: Proto-Board

Future Improvements

The current device meets all of the requirements, yet there are potential improvements to be made. Future improvements to this device could include:

Hardware Controlled Speaker Volume:
The speaker volume and frequency can be changed within the code. However, it would be convenient if the user was able to control the speaker volume without having to reprogram the microcontroller. A potentiometer could be used to control the voltage to a pin on the MSP430, which would determine the PWM and volume of the speaker output.

Test Mode for Speaker Volume:
Another convenient speaker function would be to allow the user to test the speaker volume before using the device. This would allow the user to hear the speaker volume and frequency as they control it.

Enclosure:
Because of the amount of inputs and outputs, the enclosure does not fully enclose all of the components. An improvement could be to full enclose the device, while allowing the user to have access to the inputs and outputs.

Sleep Experiments:
The REM algorithm is continuously being improved based on sleep experiments. The more sleep data that is collected, the more robust the algorithm will become. Continuing sleep experiments will test the current algorithm and reveal flaws to be improved upon.

Figure 15: PCB Enclosure

Budget

ITEM	PART#	QTY	RETAIL PRICE	AQUIRED PRICE	TOTAL
SPEAKER	CDMG16008-03-ND	1	$2.51	$2.51	$2.51
IC	ATMEGA328-PU-ND	2	$2.88	$2.88	$5.76
SPEAKER PIEZO	668-1266-ND	4	$5.38	$5.38	$21.52
Knit Conductive Silver Fabric	1167	1	$9.95	$9.95	$9.95
Stainless Thin Conductive Yarn	603	1	$4.50	$4.50	$4.50
Shipping UPS Ground	Shipping	1	$10.25	$10.25	$10.25
IC OPAMP INSTR	INA128P-ND	4	$11.31	$11.31	$45.24
LAUNCHPAD DEV BRD	296-27570-ND	2	$10.37	$10.37	$20.74
Microcontrollers	595-MSP430G2553IN20	5	$2.43	$2.43	$12.15
SWITCH PUSHBUTTON DPDT	CKN10533CT-ND	1	$1.33	$1.33	$1.33
SWITCH PUSH DPDT	P13353S-ND	2	$1.68	$1.68	$3.36
CAP CER 1UF	1276-1010-1-ND	3	$0.10	$0.10	$0.30
CAP CER 0.22UF	PCC2269CT-ND	11	$0.20	$0.20	$2.20
RES 1M OHM	P1.0MGCT-ND	3	$0.10	$0.10	$0.30
RES 5.6K OHM	P5.6KGCT-ND	2	$0.10	$0.10	$0.20
IC OPAMP INSTR	INA126P-ND	3	$4.14	$4.14	$12.42
IC OPAMP INSTR	INA126U-ND	1	$4.40	$4.40	$4.40
LED RED	511-1274-1-ND	1	$0.62	$0.62	$0.62
LED GRN	511-1272-1-ND	1	$0.62	$0.62	$0.62
TRANS NPN	MMBT3904-FDICT-ND	2	$0.12	$0.12	$0.24
SWITCH TACTILE SPST	401-1705-1-ND	1	$1.36	$1.36	$1.36
ITEM	**PART#**	**QTY**	**RETAIL PRICE**	**AQUIRED PRICE**	**TOTAL**
CAP CER 0.027U	399-3016-1-ND	2	$0.27	$0.27	$0.54
IC OPAMP INST	INA128U-ND	1	$11.70	$11.70	$11.70
RES 3.3K OHM	P3.3KGCT-ND	2	$0.10	$0.10	$0.20
RES 2K OHM	P2.0KGCT-ND	1	$0.10	$0.10	$0.10
RES 1.00K OHM	A106049CT-ND	4	$0.10	$0.10	$0.40
RES 200K OHM	P200KGCT-ND	1	$0.10	$0.10	$0.10
RES 51K OM	P51KGCT-ND	1	$0.10	$0.10	$0.10
RES 102K OHM	P102KHCT-ND	2	$0.10	$0.10	$0.20
RES 2.37K OHM	P2.37KHCT-ND	1	$0.10	$0.10	$0.10
RES 21K OHM	P21.0KHCT-ND	4	$0.10	$0.10	$0.40
RES 680 OM	P680GCT-ND	1	$0.10	$0.10	$0.10
IC OPAMP GP	MCP602-I/SN-ND	3	$0.68	$0.68	$2.04
SWITCH TACTILE SPST	401-1705-1-ND	1	$1.36	$1.36	$1.36
LE PURPLE	67-2064-ND	2	$1.22	$1.22	$2.44
LED TURQUOIS	67-2063-ND	1	$1.08	$1.08	$1.08

3-D PRINTING	-	1	$55.38	$55.38	$55.38
PCB	887212	1	$51.55	$51.55	$51.55
Bluetooth	765-RN-42	1	$17.65	$17.65	$17.65
Microcontrollers	595-MSP430G2553IN20	5	$2.43	$2.43	$12.15
Shipping	Shipping	1	$18.55	$18.55	$18.55
PCB Fabrication	887212	1	$33.00	$33.00	$33.00
				TOTAL:	**$287.76**

TOTAL FOR PCB

PCB

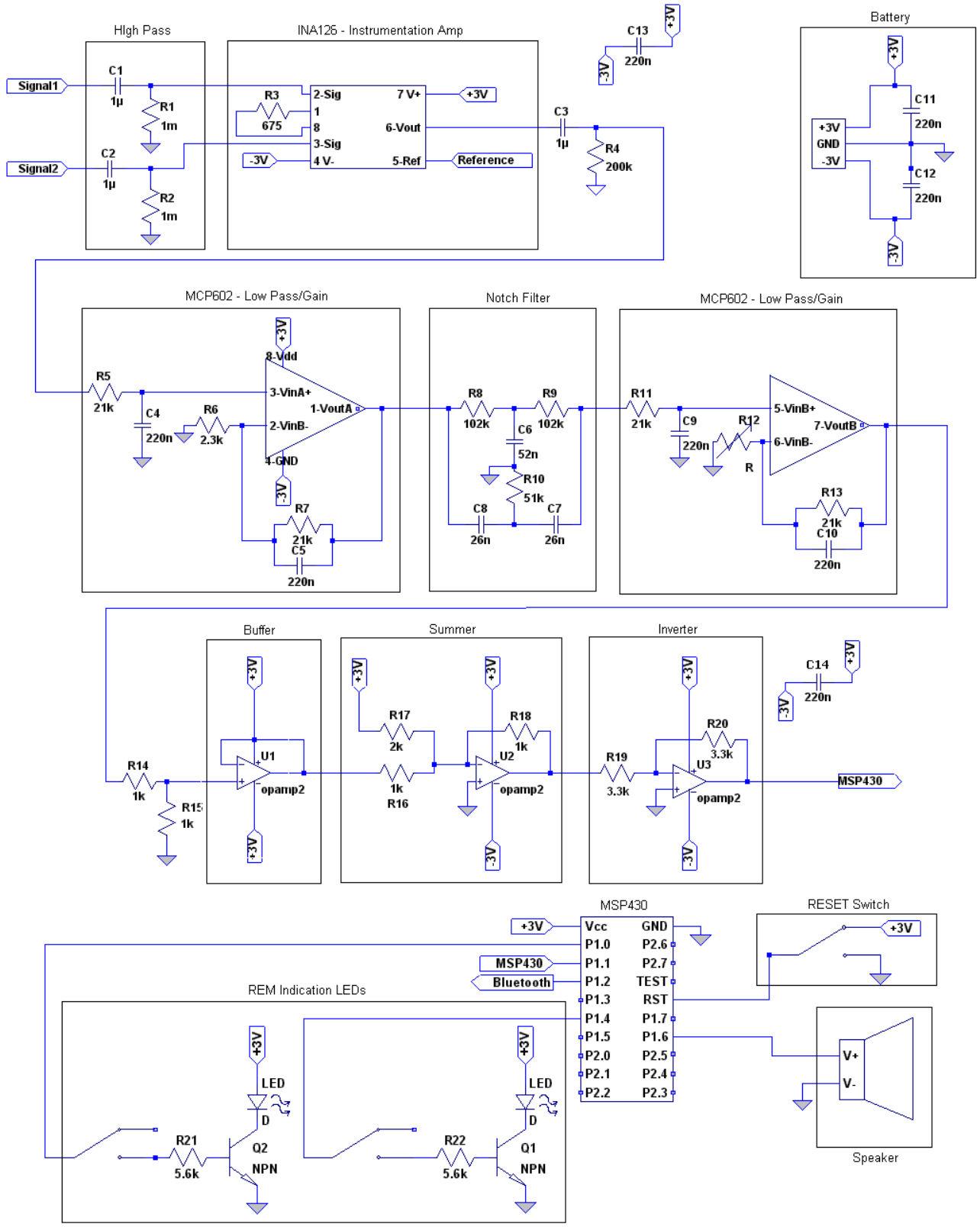
DESC	PART#	QTY	PRICE	AMT
SPEAKER PIEZO	668-1266-ND	1	$5.38	$5.38
Microcontroller	595-MSP430G2553IN20	1	$2.43	$2.43
Bluetooth	765-RN-42	1	$17.65	$17.65
IC SOCKET STRAIGHT 20POS TIN	AE9998-ND	1	$0.29	$0.29
SWITCH PUSHBUTTON DPDT 0.2A 14V	CKN10533CT-ND	1	$1.33	$1.33
SWITCH PUSH DPDT 0.2A 14V	P13353S-ND	1	$1.68	$1.68
IC OPAMP INSTR 200KHZ 8SOIC	INA126U-ND	1	$4.40	$4.40
LED 3X2MM 650NM RED WTR CLR SMD	511-1274-1-ND	1	$0.62	$0.62
LED 3X2MM 570NM GRN WTR CLR SMD	511-1272-1-ND	1	$0.62	$0.62
TRANS NPN 40V 350MW SMD SOT23-3	MMBT3904-FDICT-ND	2	$0.12	$0.24
OPAMP	MCP602-I/SN-ND	3	$0.68	$2.04
POTENTIOMETER	490-2925-ND	1	$1.51	$1.51
			TOTAL:	$38.19

TOTAL FOR PROTO-BOARD

Proto-Board

DESC	PART#	QTY	PRICE	AMT
LED 5MM 2200MCD TURQUOIS WTR CLR	67-2063-ND	1	$1.08	$1.08
LED 5MM 2200MCD PURPLE WTR CLEAR	67-2064-ND	1	$1.22	$1.22
IC OPAMP INSTR 200KHZ 8DIP	INA126P-ND	1	$4.14	$4.14
NPN TRANSISTOR	KSP10TATB-ND	2	$0.21	$0.42
SPEAKER PIEZO	668-1266-ND	1	$5.38	$5.38
Microcontroller	595-MSP430G2553IN20	1	$2.43	$2.43
Bluetooth	765-RN-42	1	$17.65	$17.65
SWITCH PUSHBUTTON DPDT 0.2A 14V	CKN10533CT-ND	1	$1.33	$1.33
SWITCH PUSH DPDT 0.2A 14V	P13353S-ND	2	$1.68	$3.36
OP AMP	MCP602-I/P-ND	3	$0.68	$2.04
POTENTIOMETER	490-2925-ND	1	$1.51	$1.51
			TOTAL:	$40.56

Appendix A: Schematic



Appendix B: PCB Layout

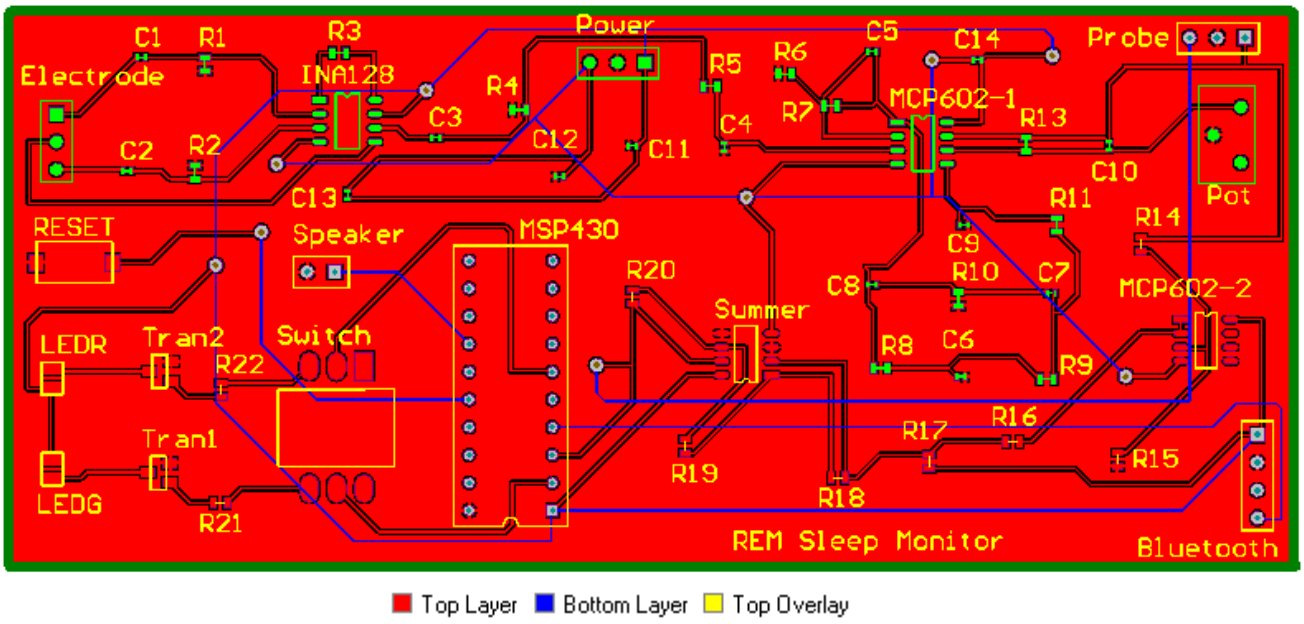


Figure x: PCB Layout

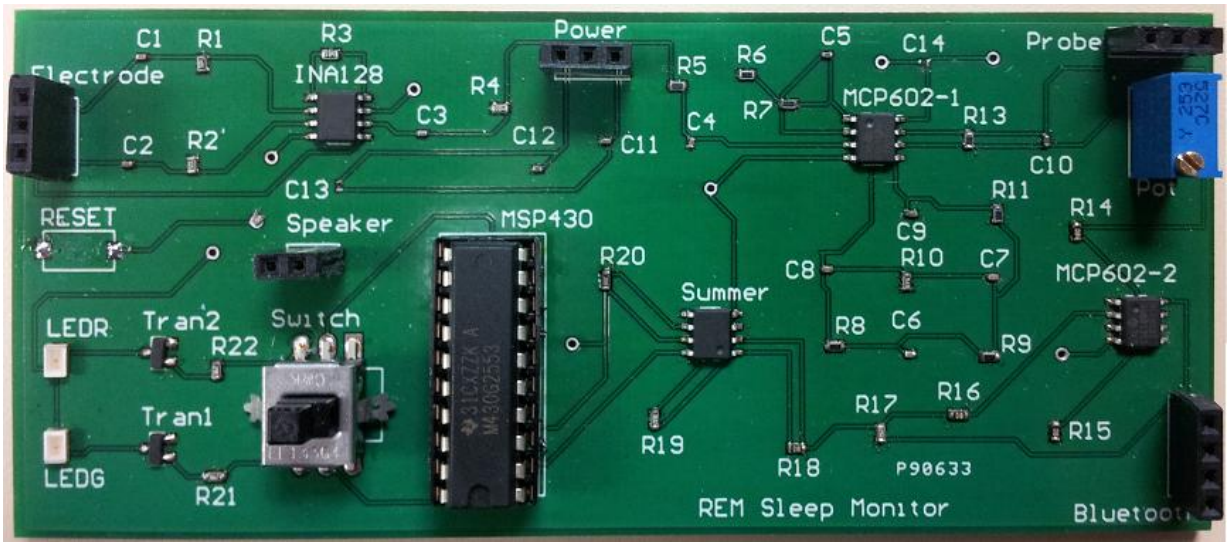


Figure x: Printed Circuit Board (PCB)

Appendix C: Bill of Materials

TOTAL FOR PCB

PCB

DESC	PART#	QTY	PRICE	AMT
SPEAKER PIEZO	668-1266-ND	1	$5.38	$5.38
Microcontroller	595-MSP430G2553IN20	1	$2.43	$2.43
Bluetooth	765-RN-42	1	$17.65	$17.65
IC SOCKET STRAIGHT 20POS TIN	AE9998-ND	1	$0.29	$0.29
SWITCH PUSHBUTTON DPDT 0.2A 14V	CKN10533CT-ND	1	$1.33	$1.33
SWITCH PUSH DPDT 0.2A 14V	P13353S-ND	1	$1.68	$1.68
IC OPAMP INSTR 200KHZ 8SOIC	INA126U-ND	1	$4.40	$4.40
LED 3X2MM 650NM RED WTR CLR SMD	511-1274-1-ND	1	$0.62	$0.62
LED 3X2MM 570NM GRN WTR CLR SMD	511-1272-1-ND	1	$0.62	$0.62
TRANS NPN 40V 350MW SMD SOT23-3	MMBT3904-FDICT-ND	2	$0.12	$0.24
OPAMP	MCP602-I/SN-ND	3	$0.68	$2.04
POTENTIOMETER	490-2925-ND	1	$1.51	$1.51
			TOTAL:	$38.19

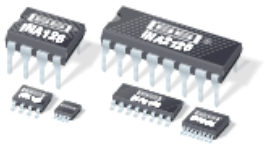
TOTAL FOR PROTO-BOARD

Proto-Board

DESC	PART#	QTY	PRICE	AMT
LED 5MM 2200MCD TURQUOIS WTR CLR	67-2063-ND	1	$1.08	$1.08
LED 5MM 2200MCD PURPLE WTR CLEAR	67-2064-ND	1	$1.22	$1.22
IC OPAMP INSTR 200KHZ 8DIP	INA126P-ND	1	$4.14	$4.14
NPN TRANSISTOR	KSP10TATB-ND	2	$0.21	$0.42
SPEAKER PIEZO	668-1266-ND	1	$5.38	$5.38
Microcontroller	595-MSP430G2553IN20	1	$2.43	$2.43
Bluetooth	765-RN-42	1	$17.65	$17.65
SWITCH PUSHBUTTON DPDT 0.2A 14V	CKN10533CT-ND	1	$1.33	$1.33
SWITCH PUSH DPDT 0.2A 14V	P13353S-ND	2	$1.68	$3.36
OP AMP	MCP602-I/P-ND	3	$0.68	$2.04
POTENTIOMETER	490-2925-ND	1	$1.51	$1.51
			TOTAL:	$40.56

Appendix D: Datasheets

Burr-Brown Products from Texas Instruments



INA126
INA2126

SBOS062A – JANUARY 1996 – REVISED AUGUST 2005

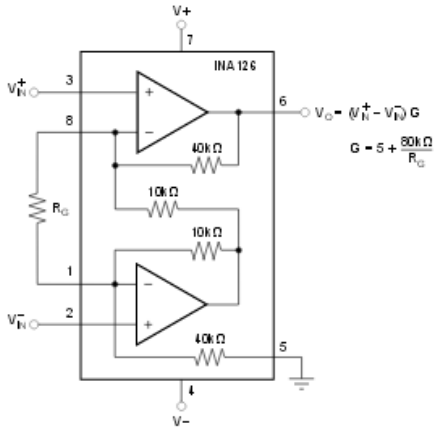
MicroPOWER INSTRUMENTATION AMPLIFIER
Single and Dual Versions

FEATURES

- LOW QUIESCENT CURRENT: 175µA/chan.
- WIDE SUPPLY RANGE: ±1.35V to ±18V
- LOW OFFSET VOLTAGE: 250µV max
- LOW OFFSET DRIFT: 3µV/°C max
- LOW NOISE: 35nV/√Hz
- LOW INPUT BIAS CURRENT: 25nA max
- 8-PIN DIP, SO-8, MSOP-8 SURFACE-MOUNT
- DUAL: 16-Pin DIP, SO-16, SSOP-16

APPLICATIONS

- INDUSTRIAL SENSOR AMPLIFIER: Bridge, RTD, Thermocouple
- PHYSIOLOGICAL AMPLIFIER: ECG, EEG, EMG
- MULTI-CHANNEL DATA ACQUISITION
- PORTABLE, BATTERY OPERATED SYSTEMS

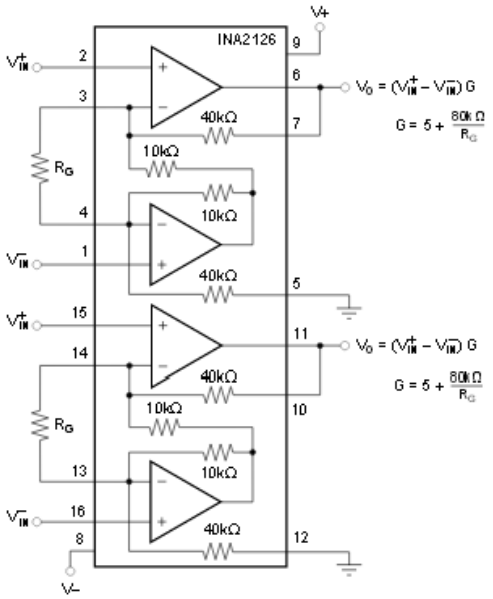


DESCRIPTION

The INA126 and INA2126 are precision instrumentation amplifiers for accurate, low noise differential signal acquisition. Their two-op-amp design provides excellent performance with very low quiescent current (175µA/channel). This, combined with a wide operating voltage range of ±1.35V to ±18V, makes them ideal for portable instrumentation and data acquisition systems.

Gain can be set from 5V/V to 10000V/V with a single external resistor. Laser trimmed input circuitry provides low offset voltage (250µV max), low offset voltage drift (3µV/°C max) and excellent common-mode rejection.

Single version package options include 8-pin plastic DIP, SO-8 surface mount, and fine-pitch MSOP-8 surface-mount. Dual version is available in the space-saving SSOP-16 fine-pitch surface mount, SO-16, and 16-pin DIP. All are specified for the –40°C to +85°C industrial temperature range.



TEXAS INSTRUMENTS
www.ti.com

MIXED SIGNAL MICROCONTROLLER

FEATURES

- Low Supply-Voltage Range: 1.8 V to 3.6 V
- Ultra-Low Power Consumption
  - Active Mode: 230 µA at 1 MHz, 2.2 V
  - Standby Mode: 0.5 µA
  - Off Mode (RAM Retention): 0.1 µA
- Five Power-Saving Modes
- Ultra-Fast Wake-Up From Standby Mode in Less Than 1 µs
- 16-Bit RISC Architecture, 62.5-ns Instruction Cycle Time
- Basic Clock Module Configurations
  - Internal Frequencies up to 16 MHz With Four Calibrated Frequency
  - Internal Very-Low-Power Low-Frequency (LF) Oscillator
  - 32-kHz Crystal
  - External Digital Clock Source
- Two 16-Bit Timer_A With Three Capture/Compare Registers
- Up to 24 Capacitive-Touch Enabled I/O Pins
- Universal Serial Communication Interface (USCI)
  - Enhanced UART Supporting Auto Baudrate Detection (LIN)
  - IrDA Encoder and Decoder
  - Synchronous SPI
  - I²C™
- On-Chip Comparator for Analog Signal Compare Function or Slope Analog-to-Digital (A/D) Conversion
- 10-Bit 200-ksps Analog-to-Digital (A/D) Converter With Internal Reference, Sample-and-Hold, and Autoscan (See Table 1)
- Brownout Detector
- Serial Onboard Programming, No External Programming Voltage Needed, Programmable Code Protection by Security Fuse
- On-Chip Emulation Logic With Spy-Bi-Wire Interface
- Family Members are Summarized in Table 1
- Package Options
  - TSSOP: 20 Pin, 28 Pin
  - PDIP: 20 Pin
  - QFN: 32 Pin
- For Complete Module Descriptions, See the *MSP430x2xx Family User's Guide* (SLAU144)

DESCRIPTION

The Texas Instruments MSP430 family of ultra-low-power microcontrollers consists of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low-power modes, is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 1 µs.

The MSP430G2x13 and MSP430G2x53 series are ultra-low-power mixed signal microcontrollers with built-in 16-bit timers, up to 24 I/O capacitive-touch enabled pins, a versatile analog comparator, and built-in communication capability using the universal serial communication interface. In addition the MSP430G2x53 family members have a 10-bit analog-to-digital (A/D) converter. For configuration details see Table 1.

Typical applications include low-cost sensor systems that capture analog signals, convert them to digital values, and then process the data for display or for transmission to a host system.

Appendix E: Software Code

```
*By James Parrow
 * REV. 2.7
 * November 15, 2013

                       MSP430G2553
                     -----------------
               3v  -->|Vcc         Vss|<--  GRN
     REM Detection LED  <--|P1.0           |
      ADC (Electrode)  -->|P1.1           |
          UART (RX)  <--|P1.2           |
                     |           P1.6|-->  PWM (Speaker)
   REM Not Detected LED  <--|P1.4           |

The goal of this project is to induce lucid dreaming

This project will receive a analog signals from electrodes placed on a human. After receiving
the analog signal the data will be sent via UART to a bluetooth device. The data will be also
used to compare the signal strength to that of REM Sleep. After REM Sleep is detected for a
chosen period of time a speaker will sound at low frequency, to quo the user that they are
dreaming. After a chosen time, the MSP430 will change the speaker frequency and duty cycle
to wake the user.
 */

#include "msp430g2553.h"

int timerCount = 0;
signed int REM = 0;
int TimeTrack = 0;
int TimeM= 0;

//////////////////// Initialize variables below to desired value ///////////////////////

int ScanPeriodInREM = 457;     // 457 is approx 30 seconds
int NumberOfScanPeriods = 10;   // 10 periods is approx 5 mins total time scanning
int MinsInLowFre = 5;          // 5 is approx 5 mins that Low Frequency speaker is sounded (this is the lucid dream cue)

//////////////////// Initialize variables above to desired value ///////////////////////




////////////////////////////////////////////////Includes above and int above//////////////////

void main(void)
{
            ////////////////////////////////ADC Set-Up ////////////////////////////////////////////
            WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
            P1OUT |= 8;                     // writes a value on the output pins 8 in Dec = 1000, Same as 1<<3, P1.3
            P1REN |= 8;                     // Enable internal pull up  8 in Dec = 1000, Same as 1<<3, P1.3
            P1DIR |= 1;                    // Set P1.0 to output direction
            P1DIR |= 1<<4;                  // Set P1.4 to output direction
            P1DIR |= 1<<6;                  // Set P1.6 to output direction
            P1OUT &= ~(1+ (1<<4) + (1<<6));  // Set the LEDs off (P1.0, P1.4, P1.6 set off)
            P1IE |= 0x08;                  // P1.3 interrupt enabled
            P1IFG &= ~0x08;                 // P1.3 IFG cleared
            CCTL0 = CCIE;                  // CCR0 interrupt enabled
            TACTL = TASSEL_2 + MC_2;        // Set the timer A to SMCLCK, Continuous
```

```
        __enable_interrupt();


        ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE;  // ADC10ON, interrupt enabled
        ADC10CTL1 = INCH_1;                                  // input A1
        ADC10AE0 |= 0x02;                  // PA.1 ADC option select (ADc is on P1.1)

  //Calibrate DCO
   BCSCTL1 = CALBC1_1MHZ;                // DCO at 1 MHz
   DCOCTL = CALDCO_1MHZ;                 // DCO at 1 MHz

  // UART Set-Up
   P1SEL |= BIT2;                  // Use P1.2 as USCI_A0
   P1SEL2|= BIT2;                   // Use P1.2 as USCI_A0
   P1DIR |= BIT2;                  // Set 1.2 as output
   UCA0CTL1 |= UCSSEL_2;              // Use SMCLK / DCO
   UCA0BR0 = 104;                    // 1 MHz -> 9600   N=Clock/Baud
   UCA0BR1 = 0;                    // 1 MHz -> 9600
   UCA0MCTL = UCBRS1;                // Modulation UCBRSx = 1
   UCA0CTL1 &= ~UCSWRST;               // **Initialize USCI

   //General Interrupt Enable
    _BIS_SR(GIE);

   // _BIS_SR(LPM0_bits + GIE);          // Enter LPM3 // Exit is located in interrupt, not efficient with no crystal and ADC interrupt

    for (;;)
          {
                ADC10CTL0 |= ENC + ADC10SC;             //Sampling and conversion start
                // __vis_SR_register(CPUOFF + GIE);                  // LPM0, ADC10_ISR will force exit (does not always Exit ==
Code No work)

                if (ADC10MEM < 0x266)                  //Low REM Voltage (1.8V is where REM starts, REM is between 1.8 and
2.8V)
                {
                      P1OUT &= ~0x01;                        //Clear P1.0 LED off
                      P1OUT |= 0x10;                         //Set P1.4 LED on
                }
                else if (ADC10MEM > 0x3BB)  // High (over 2.8V is saturated)
                {
                      P1OUT |= 0x10;                         //Set P1.4 LED on
                      P1OUT &= ~0x01;                        //Clear P1.0 LED off
                      REM--;                                       // Subtracts to REM if in voltage range
                }
                else
                {
                      P1OUT |= 0x01;                         //Set P1.0 LED on
                      P1OUT &= ~0x10;                        //Clear P1.4 LED off
                      REM++;                                       // Adds to REM if in voltage range
                }



                if((timerCount==ScanPeriodInREM) && (REM>1))   //statement makes the scan period for REM
                {
                      //P1OUT ^= 1<<6;    // P1.6 = toggle// Used for Debug not needed
                      TimeTrack++;      // adds 30 second to the timer every time
                      timerCount=0;     //resets clock
                      REM=0;          // reset eye movement
                }
```

```
                ////////////////////above Reads REM movement and increments TimerTrack if REM movement is detected
///////////////

                if(timerCount==(ScanPeriodInREM+2))     // If REM is not detected Statement is entered
                {
                        TimeTrack=0;
                        timerCount=0;
                        REM=0;
                }                       // Statement is used to reset if REM is not detected

                ///////////////above if REM is not detected in 30 seconds then REM detections is started from
beginning//////////

                if((TimeTrack==NumberOfScanPeriods)  && (timerCount<ScanPeriodInREM)) //Statement is entered if all scan
periods are met
                {
                        //P1OUT ^= 1<<6;    // P1.6 = toggle// Used for Debug not needed
                        timerCount=460;     // timerCount is set to 460 to stay out of all statements above
                        TimeTrack=0;
                }

                if((timerCount>465))          // For this statement timer is sped up, 1second = 500 timerCount
                {
                        P1DIR |= (1<<6);                        //P1.6 output
                        P1SEL |= (1<<6);                        //P1.6 TA1/2 options
                        CCR0 = 1000;                              // PWM Period/2
                        CCTL1 = OUTMOD_6;                         // CCR! toggle/set
                        CCR1 = 5;                                 // CCR1 PWM duty cycle
                        TACTL = TASSEL_2 + MC_3;       // SMCLK, up-down mode
                        if(timerCount>30466)
                        {
                                TimeM = TimeM +1;


                                while(TimeM==MinsInLowFre)     // For this statement timer is sped up,  1second = 500
timerCount
                                {
                                        P1DIR |= (1<<6);                    //P1.6 output
                                        P1SEL |= (1<<6);                      //P1.6 TA1/2 options
                                        CCR0 = 100;                               // PWM Period/2
                                        CCTL1 = OUTMOD_6;                     // CCR! toggle/set
                                        CCR1 = 50;                                              // CCR1 PWM duty cycle
                                        TACTL = TASSEL_2 + MC_3;       // SMCLK, up-down mode
                                        TimeM=MinsInLowFre;
                                }                       //CODE IS STUCK HERE, hard reset required to use REM Monitor again

                                timerCount=466;             // timerCount is set to 466 to stay in this statement
                        }
                }

        }
}

///////////////Main above, set timers pins interrupts ///////////////////////
```

```
// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
	timerCount = (timerCount + 1); // Clock at 1Mh (time per clock =65536/1000000)
	//////////////////////Increments timer above//////////////////////////
	if(timerCount%2)	// Send data every other timerCount, (too much data slows Labview)
	UCA0TXBUF = ADC10MEM;	// UART stored and sent in UCA0TXBUF
/*
	 if(timerCount>18280) // 18280 is approx 20 min, MSP430 is in sleep mode for 20 min // Used for Low power mode, With
ADC, interrupt, and no crystal Not Efficient
	{
		_BIC_SR_IRQ(LPM0_bits);		// Clear LPM3 bits from 0(SR)
	}
*/
}



	// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
	__bic_SR_register_on_exit(CPUOFF);			// Clear CPUOFF bit from 0 (SR)
	// __delay_cycles(500); If adc acts to fast, delay can be used
}
```